The first code merges the translation to Normalized Low Pass (NLP) and the design equations together

```
function n = bwlpdsgn(wc,ws,atten)
%
% n = bwlpdsgn(wc,ws,atten)
%
%   This function will calculate the order of Butterworth filter required
% to meet the Low-Pass specification.
% The specification is given as
%      a cut-off frequency (wc in rad/sec),
%      a stop frequency (ws in rad/sec)
%  and a maximum attenuation in the stop-band (atten in dB).

n = ceil(atten/(-20*log10(ws/wc)));

return;
```

And

```
function n = bwbpdsgn(wsl,wcl,wcu,wsu,atten)
%
% n = bwbpdsgn(wsl,wcl,wcu,wsu,atten);
%
%   This function will calculate the order of Butterworth filter
% required to meet a Band-Pass specification. The specification is given
% a as
%     lower stop frequency (wsl - in rad/sec),
%     lower cut-off frequency (wcl in rad/sec),
%     upper cut-off frequency (wcu in rad/sec),
%     upper stop frequency (wsu in rad/sec)
% and a maximum attenuation in the stop-band (atten in dB).
%

su = abs((wcl*wcu-wsu*wsu)/(wsu*abs(wcu-wcl)));
sl = abs((wcl*wcu-wsl*wsl)/(wsl*abs(wcu-wcl)));

su = min(su,sl);

n = ceil(atten/(-20*log10(su)));

return;
```

And

```
function n = bwhpdsgn(wc,ws,atten)
%
%  n = bwhpdsgn(wc,ws,atten)
%
%   This function will calculate the order of Butterworth filter required
% to meet the High-Pass specification.
% The specification is given as
%       a cut-off frequency (wc in rad/sec),
%       a stop frequency (ws in rad/sec)
%   and a maximum attenuation in the stop-band (atten in dB).

n = ceil(atten/(-20*log10(wc/ws)));

return;
```

A similar set for Chebyshev software

```
function [epsilon,n] = chbylpds(wc,rip,ws,atten)
%
% [epsilon,n] = chbylpds(wc,rip,ws,atten)
%
%   This function will calculate the epsilon and order of Chebyshev filter
% required to meet the Low-Pass specification.
% The specification is given as
%       a cut off frequency (wc in rad/sec),
%       an allowable ripple in the pass-band (rip in dB),
%       a stop frequency (ws in rad/sec)
%   and a maximum attenuation in the stop-band (atten in dB).
epsilon = sqrt(10^(rip/10)-1);
n = ceil(acosh(sqrt(10^(abs(atten)/10)-1)/epsilon)/acosh(ws/wc));
return;
```

And

```
function [epsilon,n] = chbybpds(ripple,wsl,wcl,wcu,wsu,atten)
%
% [epsilon,n] = chbybpds(ripple,wsl,wcl,wcu,wsu,atten);
%
%   This function will calculate the epsilon and order of Chebyshev filter
% required to meet a Band-Pass specification. The specification is given
% a as
%     allowable ripple in the pass-band (rip in dB),
%     lower stop frequency (wsl - in rad/sec),
%     lower cut-off frequency (wcl in rad/sec),
%     upper cut-off frequency (wcu in rad/sec),
%     upper stop frequency (wsu in rad/sec)
% and a maximum attenuation in the stop-band (atten in dB).
su = abs((wcl*wcu-wsu*wsu)/(wsu*abs(wcu-wcl)));
sl = abs((wcl*wcu-wsl*wsl)/(wsl*abs(wcu-wcl)));
su = min(su,sl);
epsilon = sqrt(10^(ripple/10)-1);
n = ceil(acosh(sqrt(10^(abs(atten)/10)-1)/epsilon)/acosh(su));

return;
```

And

```
function [epsilon,n] = chbyhpds(wc,rip,ws,atten)
%
% [epsilon,n] = chbyhpds(wc,rip,ws,atten)
%
%   This function will calculate the epsilon and order of Chebyshev filter
% required to meet the High-Pass specification.
% The specification is given as
%       a cut off frequency (wc in rad/sec),
%       an allowable ripple in the pass-band (rip in dB),
%       a stop frequency (ws in rad/sec)
%   and a maximum attenuation in the stop-band (atten in dB).

epsilon = sqrt(10^(rip/10)-1);
n = ceil(acosh(sqrt(10^(abs(atten)/10)-1)/epsilon)/acosh(wc/ws));
return;
```

Now we have code that will create the H(s) for each type

```
function a = bw_hs(n)
%
%  a = bw_hs(n);
%
% This function will generate an array of coefficients
% which make up the H(s) for an nth order Butterworth filter.
% The coefficients (a) are in a 3 x m array, where m = int( (n+1)/2 ).

for k=1:n/2,
  b = pmulti(pmulti(1,exp(-j*pi*(2*k+n-1)/(2*n))) ...
                     ,exp( j*pi*(2*k+n-1)/(2*n)));
  % trim off any imaginary parts ( should only be roundoff error ).
  a(k,:) = real(b);
end;

% insert Odd order single pole.
if rem(n,2)>eps,
 a((n+1)/2,:) = [0 1 1];
end;
return;


function [a,gain] = chby_hs(e,n)
%
%  [a,gain] = chby_hs(e,n);
%
% This function will generate an array of coefficients and a gain term
% which make up the H(s) for an nth order Chebyshev filter
% with e setting the ripple.

b(1) = 1;
a2 = (sqrt(1+(1/e^2))+1/e)^(1/n);
a1 = 0.5*(a2-1/a2);
b1 = 0.5*(a2+1/a2);
a2 = 1;
for k=1:n/2,
 bw = exp(j*pi*(2*k+n-1)/(2*n));
 b = pmulti(pmulti(1,a1*real(bw)+j*b1*imag(bw))  ...
                    ,a1*real(bw)-j*b1*imag(bw));
 a(k,:) = real(b);
 a2 = a2*a(k,3);
end;
gain = 1/sqrt(1+e*e);
if rem(n,2)>eps,
  a((n+1)/2,:) = [0 1 a1];
  gain = 1;
  a2 = a2*a1;
end;
gain = gain*a2;
return;
```

Then a set translation code going from NLP to the desired type

```
function [num,denom] = nlp_lp(a,gain,wc);
%
% [num,denom] = nlp_lp(a,gain,wc)
%
%   This function will translate the NLP filter given by "a" and "gain"
% into a Low-Pass filter with a cutoff of wc.  For Butterworth gain = 1.
%

[n,m] = size(a);

if m~=3,
  disp('The input a is not a valid H(s) array.');
  return;
end;

for k=1:n,
  num(k,:) = [0 0 1];
  denom(k,:) = [a(k,1)/(wc*wc) a(k,2)/wc a(k,3)];
end;

num(1,3) = gain;

return;

function [num,denom] = nlp_hp(a,gain,wc);
%
% [num,denom] = nlp_hp(a,gain,wc);
%
%   This function will translate the NLP filter given by "a" and "gain"
% into a HP filter with a cutoff of wc.  For Butterworth gain = 1.
%

[n,m] = size(a);

if m~=3,
  disp('The input a is not a valid H(s) array.');
  return;
end;

for k=1:n,
  if a(k,1)>eps,
    num(k,:) = [1/(wc*wc) 0 0];
    denom(k,:) = [a(k,3)/(wc*wc) a(k,2)/wc a(k,1)];
  else
    num(k,:) = [0 1/wc 0];
    denom(k,:) = [0 a(k,3)/wc a(k,2)];
  end;
end;

if a(1,1)>eps,
  num(1,1) = gain*num(1,1);
else
  num(1,2) = gain*num(1,2);
end;

return;
```

```matlab
function [num,denom] = nlp_bp(b,gain,wcl,wcu)
%
% [num,denom] = nlp_bp(a,gain,wcl,wcu)
%
%   This function will translate the NLP filter given by "a" and "gain"
% into a BP filter with a pass band of wcl to wcu.
% For Butterworth gain is equal to 1.
%

[n,m] = size(b);
if abs(b(n,1)) > eps,
  num   = zeros( 2*n  , m );
  denom = zeros( 2*n  , m );
else
  num   = zeros( 2*n-1, m );
  denom = zeros( 2*n-1, m );
end;
wc = sqrt(abs(wcl*wcu));
wb = abs( max(wcu,wcl) - min(wcu,wcl) );

g = 1.0;
for k=1:n,
  if abs( b(k,1) ) > eps,
     d(5) = 1;
     d(4) = b(k,2)*wb/(wc*wc*b(k,1));
     d(3) = (2/(wc^2) + b(k,3)*wb*wb/(b(k,1)*wc^4));
     d(2) = b(k,2)*wb/(b(k,1)*wc^4);
     d(1) = 1/(wc^4);
     rd = roots(d);

     denom( 2*k-1, : ) = pmulti(pmulti(1,rd(1)),rd(2));
     num(   2*k-1, 3 ) = 1/b(k,1);

     g = g *            num(2*k-1,3)                              ...
           / sqrt(  (denom(2*k-1,3)-denom(2*k-1,1)*wc*wc)^2 ...
                  + (denom(2*k-1,2)*wc)^2                     );
     denom( 2*k, : )   = pmulti(pmulti(1,rd(3)),rd(4));
     num(   2*k, 1 )   = wb*wb/(wc^4);
     g = g *             num(2*k,1)  * wc * wc                 ...
           / sqrt(   (denom(2*k,3)-denom(2*k,1)*wc*wc)^2 ...
                   + (denom(2*k,2)*wc)^2                     );
  else
     denom( 2*k - 1, 1 ) = b(k,2);
     denom( 2*k - 1, 2 ) = wb*b(k,3);
     denom( 2*k - 1, 3 ) = wc*wc*b(k,2);
     num(   2*k - 1, 2 ) = wb;
     g = g *             num(2*k-1,2)  * wc                       ...
           / sqrt(   (denom(2*k-1,3)-denom(2*k-1,1)*wc*wc)^2 ...
                   + (denom(2*k-1,2)*wc)^2                       );
  end;
end;

if abs(num(1,3)) > eps,
  num(1,3) = num(1,3)*gain/g;
else;
  num(1,2) = num(1,2)*gain/g;
end;

return;
```

If we are going to work these as digital filters we have a bilinear transform to convert the H(s) to an H(z)

```
function [num,denom] = hs_z(b,a)
%
% usage: [num,denom] = hs_z(b,a)
%
% This function is to translate an H(s) to an H(z)
% using the bilinear transformation.
% Input:
%   numerator given by b (an M by 3 matrix),
%   denominator given by a (an N by 3 matrix).
%   These can be generated by the
%   functions chby_hs and bw_hs.
% Output:
%   numerator given by num (an M by 3 matrix) and
%   denominator given by denom (an N by 3 matrix).
%   These can be used with hzval to generate a
%   frequency plot.
%

[n,m]   = size(a);
[n1,m1] = size(b);

if m~=3|m1~=3,
  disp('These are not valid H(s) arrays.');
  return;
end;
if n~=n1,
  disp('The input numerator and denominator do not match up.');
  return;
end;

for k=1:n,
  denom(k,:) = [  (a(k,1)+a(k,2)+a(k,3)) -2*(a(k,1)-a(k,3)) ...
                  (a(k,1)-a(k,2)+a(k,3)) ];
  num(k,:)   = [  (b(k,1)+b(k,2)+b(k,3)) -2*(b(k,1)-b(k,3)) ...
                  (b(k,1)-b(k,2)+b(k,3)) ];
end;

return;
end;
```

And to support functions, the first is simply a function to evaluate the H(z) in the format of second order pairs.

```
function g = hzval(num,denom,z)
%
% g = hzval(num,denom,z);
%
[n,m] = size(num);
g = 1;
for k=1:n,
  g = g .* (z.*z.*num(k,1)   + z.*num(k,2)   + num(k,3)   ) ./ ...
           (z.*z.*denom(k,1) + z.*denom(k,2) + denom(k,3));
end;
return;
```

And the other will implement the H(z), applying it to a signal.

```
function g = implement_hz(num,denom,x)
%
% g = implement_hz(num,denom,x);
%

[n,m] = size(num);
y = x;
for k=1:n,
   g = second_order_Hz( y, num(k,:), denom(k,:) );
   y = g;
end;
return;

function z = second_order_Hz( x, b, a )

[len,width] = size(x);

if( abs( a(1) ) > eps )
   y  = b(1)/a(1) * x(3:len) ...
      + b(2)/a(1) * x(2:len-1) ...
      + b(3)/a(1) * x(1:len-2);
   for k = 3:len-2,
      y(k) = y(k) - a(2)/a(1) * y(k-1) - a(3)/a(1) * y(k-2);
   end;
   z = y(3:len-2);
else
   y  = b(2)/a(2) * x(2:len,:) ...
      + b(3)/a(2) * x(1:len-1,:);
   for k = 2:len-1,
      y(k) = y(k) - a(3)/a(2) * y(k-1);
   end;
end;

return;
```

The following two functions are simply support for the previous functions.

```
function b = pmulti(a,z);
%
%   b = pmulti(a,z);
%
%   This function will multiply a polynomial "a" with a first-order
% polynomial described by a zeros "z".

n = length(a);
b = zeros(1,n+1);
b(1) = a(1);

for k=2:n,
  b(k) = a(k) - z*a(k-1);
end;

b(n+1) = - z*a(n);

return;
end;
```